P.Ż

WPROWADZENIE DO ODCZYTYWANIA I ANALIZY DANYCH

CO DZISIAJ – POBIERANIE TREŚCI ABY JĄ PÓŹNIEJ ANALIZOWAĆ – PODSTAWY PRZED BUDOWĄ WIĘKSZEJ SIECI. ODCZYTYWANIE TREŚCI Z RÓŻNYCH ŹRÓDEŁ

A MOŻE INACZEJ - ANALIZA TEKSTU

- Python jest językiem wykorzystywanym często we wszelakiej analizie danych
- Parsowanie tekstu w Pythonie to proces analizowania tekstu, którego celem jest zrozumienie jego struktury i znaczenia. Technika ta jest powszechnie stosowana w działaniach na danych, np. przy analizie logów, parsowaniu stron internetowych itp.

- W Pythonie istnieją różne biblioteki i moduły do parsowania tekstu, takie jak na przykład `re` (Regular Expressions) lub `BeautifulSoup`. Moduł `re` umożliwia parsowanie tekstu przy użyciu wyrażeń regularnych, zajmuje się on dopasowywaniem określonych wzorców w tekście. Natomiast `BeautifulSoup` służy do parsowania HTML i XML.

- Ogólnie rzecz biorąc, parsowanie tekstu w Pythonie polega na wyodrębnieniu określonych elementów tekstu na podstawie określonych kryteriów, np. wzorców wyrażeń regularnych. Znajomość podstawowych technik parsowania tekstu jest bardzo cenna i przydatna w różnych dziedzinach programowania.

BEAUTIFUL SOUP TRANSFORMUJE DOKUMENTY HTML DO POSTACI DRZEWA OBIEKTÓW PYTHON

Cyt za dokumentacją modułu

A MOŻE INACZEJ – ANALIZA TEKSTU

- BS4 wykorzystuje się klasycznie w połączeniu z request
- W trakcie naszej pracy, będziemy wykorzystywać bibliotekę Beautiful Soup do przetwarzania strony internetowej. Do tego celu wykorzystamy obiekt klasy BeautifulSoup, który będzie reprezentował stronę. Jako argument do konstruktora podamy tekst html strony, który wcześniej pobierzemy przy użyciu funkcji get z biblioteki requests. W drugim argumencie konstruktora podamy rodzaj parsera. Beautiful Soup to narzędzie nie tylko do parsowania HTML ale też XML. Na końcu ustalimy kodowanie w jednej z linii kodu. Przed przystąpieniem do przetwarzania strony, na sąsiedniej karcie, możemy zobaczyć kod HTML, który będziemy analizować.”

```
1  import requests as req
2  from bs4 import BeautifulSoup
```

```
import requests as req
from bs4 import BeautifulSoup
strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
strona.encoding='utf-8'
zupa=BeautifulSoup(strona.text,'html.parser')
```

A MOŻE INACZEJ – ANALIZA TEKSTU Z WWW

- Metoda ".h3" wyodrębnia z obiektu "zupa" pierwszy znaleziony element HTML o nazwie "h3".
- - Funkcja "print" wyświetla ten znaleziony element HTML, który jest obiektem klasy "Tag" w przypadku znalezienia takiego elementu na stronie.
- - Funkcja "type" zwraca typ obiektu "zupa.h3", który w tym przypadku powinien być również "Tag".
-

```
1  import requests as req
2  from bs4 import BeautifulSoup
3  strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
4  strona.encoding='utf-8'
5  zupa=BeautifulSoup(strona.text,'html.parser')
6  print(zupa.h3)
7  print(type(zupa.h3))
```

```
1  import requests as req
2  from bs4 import BeautifulSoup
3  strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
4  strona.encoding='utf-8'
5  zupa=BeautifulSoup(strona.text,'html.parser')
6  print(zupa.find('h3'))
7  print(type(zupa.find('h3')))
```

A MOŻE INACZEJ – ANALIZA TEKSTU Z WWW

- Im głębiej w strukturę strony wchodzimy, tym możemy więcej elementów używać

```python
import requests as req
from bs4 import BeautifulSoup
strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
strona.encoding='utf-8'
zupa=BeautifulSoup(strona.text,'html.parser')
print(zupa.find('h3'))
print(type(zupa.find('h3')))
```

```python
import requests as req
from bs4 import BeautifulSoup
strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
strona.encoding='utf-8'
zupa=BeautifulSoup(strona.text,'html.parser')
print(zupa.find(id='sekcja2'))
```

```
<div class="podsekcja" id="sekcja2">
        Zawartość sekcji numer 2
    </div>
```

A MOŻE INACZEJ – ANALIZA TEKSTU Z WWW

- Możliwości wyszukiwania jest więcej
-

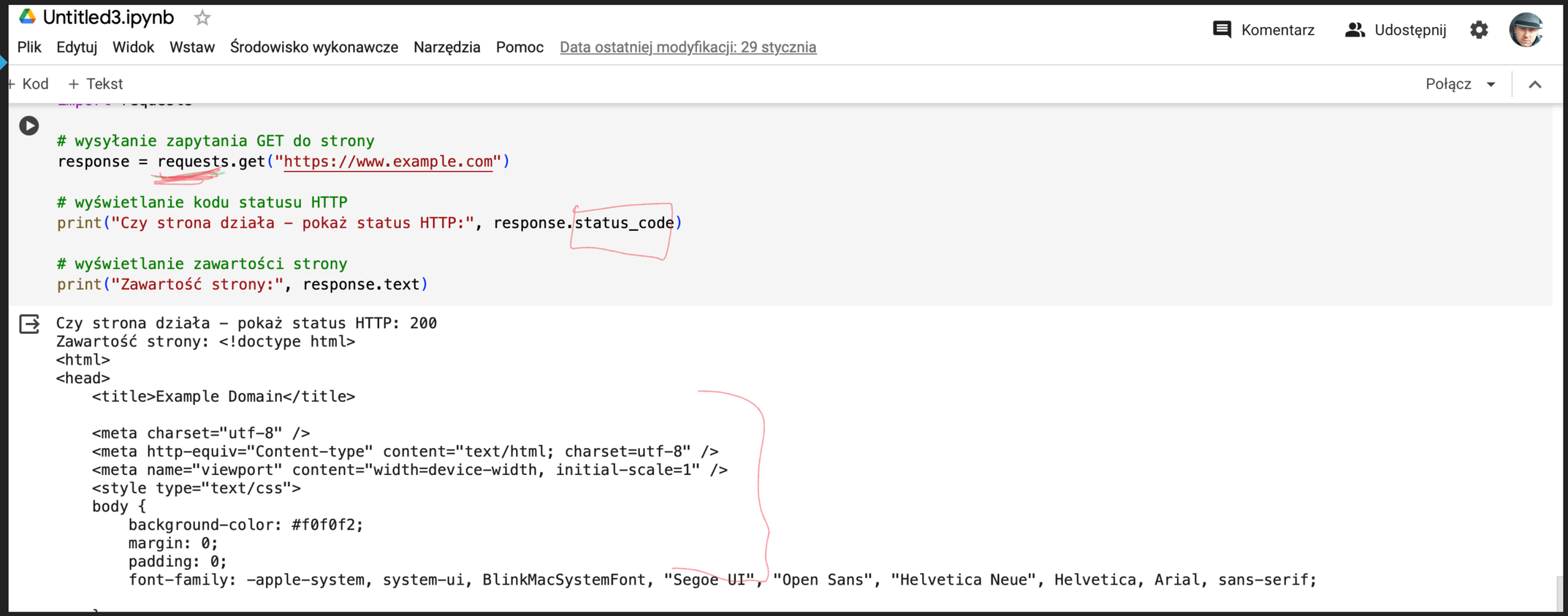
```
import requests as req
from bs4 import BeautifulSoup
strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
strona.encoding='utf-8'
zupa=BeautifulSoup(strona.text,'html.parser')
print(zupa.find(class_='podsekcja'))
```

```
import requests as req
from bs4 import BeautifulSoup
'http://jsystems.pl/static/data/pnl/dane.html')
='utf-8'
oup(strona.text,'html.parser')
(attrs={'name':'stopka'}))
```

```
import requests as req
from bs4 import BeautifulSoup
strona=req.get('http://jsystems.pl/static/data/pnl/dane.html')
strona.encoding='utf-8'
zupa=BeautifulSoup(strona.text,'html.parser')
print(zupa.title)
print(zupa.head)
print(zupa.body)
```

A MOŻE INACZEJ – ANALIZA TEKSTU Z WWW

- Możliwości wyszukiwania jest więcej



ALTERNATYWY DLA BEAUTIFUL SOUP

- 1. lxml - biblioteka do przetwarzania i parsowania XML i HTML, która oferuje opcje szybkiego przetwarzania oraz układania DOM. Posiada również wiele funkcjonalności do przeszukiwania, transformowania i walidowania dokumentów XML i HTML.

- 2. PyQuery - biblioteka oparta na jQuery, która umożliwia przeszukiwanie i manipulowanie dokumentami HTML i XML. Jest to bardzo łatwa w użyciu biblioteka, która pozwala na korzystanie z wygodnego API do wyszukiwania elementów HTML i wykorzystywania selektorów CSS.

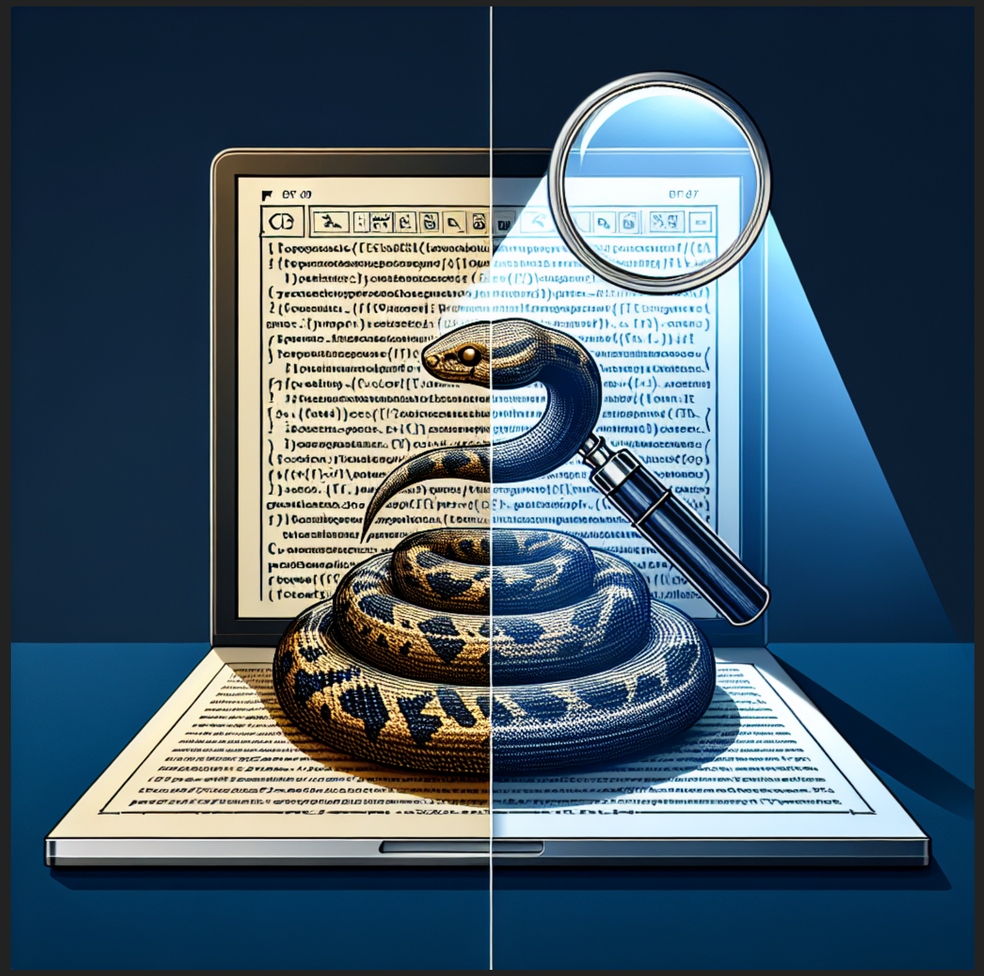
- 3. html5lib - biblioteka do przetwarzania dokumentów HTML5, która oferuje stabilne i dokładne parsowanie dokumentów. Jest ona nieco wolniejsza niż lxml, ale oferuje lepszą zgodność z HTML5 i może być wygodnym narzędziem do parsowania nieprawidłowo sformowanych dokumentów HTML.

- 4. re - biblioteka do pracy z wyrażeniami regularnymi, która może być używana do przetwarzania i parsowania tekstu HTML i XML. Jest to bardziej podstawowe podejście niż wykorzystanie biblioteki do parsowania dokumentów, ale może być przydatne w przypadku prostych zadań.

- 5. Scrapy - framework stanowiący kompletny zestaw narzędzi do przetwarzania i parsowania stron internetowych. Oferuje on elastyczne podejście do przetwarzania stron oraz bogatą funkcjonalność do przeszukiwania i pobierania danych z witryn internetowych, a także tworzenia pełnoprawnych aplikacji internetowych.

WYKORZYSTUJEMY NP. PRZY PODEJMOWANIU DECYZJI ZWIĄZANYCH Z PARKINGAMI, AUTOMATYKĄ DOMOWĄ ITP

OCR TO ROZPOZNAWANIE ZNAKÓW LUB TEKSTU Z OBRAZÓW LUB RĘCZNIE STWORZONYCH DOKUMENTÓW.



OCR TO ROZPOZNAWANIE ZNAKÓW LUB TEKSTU Z OBRAZÓW LUB RĘCZNIE STWORZONYCH DOKUMENTÓW.

Do najpopularniejszych bibliotek w python, związanych z OCR należą:

- Tesseract - pytesseract
- OCRopus
- Kraken
- easyOCR

TESSERACT

Biblioteka, która w zasadzie jest API dostarczanym przez Google. Ogromne możliwości, ale raczej w większych zastosowaniach

```python
import pytesseract
from PIL import Image

# Wczytanie obrazu
image = Image.open('tekst.jpg')
# Konwersja obrazu na skale szarosci
image = image.convert('L')

# Rozpoznawanie tekstu przy uzyciu Tesseract
text = pytesseract.image_to_string(image)

# Wypisanie rozpoznanego tekstu
print(text)
```

KRAKEN

Od podstaw zbudowana pod kątem wykorzystywania sieci neuronowych (przykład rozbudowany później - zawiera więcej na początku niż Tesseract - 0 i 1 oznaczają, że ma być sam tekst)

```python
import kraken

# Wczytanie obrazu
img = kraken.image.imread('tekst.jpg')

# Rozpoznawanie tekstu przy uzyciu Kraken
text = kraken.recognize(img)

# Wypisanie rozpoznanego tekstu
print(text[0][1])
```

EASYOCR

Szybkie i proste. Nie ma problemu z sieciami neuronowymi. Trochę słabiej działa na obrazach o niskiej jakości.

```python
import easyocr

# Wczytanie obrazu
reader = easyocr.Reader(['en'])
results = reader.readtext('tekst.jpg')

# Wypisanie rozpoznanego tekstu
for res in results:
    print(res[1])
```

TESSERACT

W bibliotece Tesseract sam proces trenowania modeli OCR na bazie sieci neuronowych nie jest łatwy, ale możemy wykorzystać już gotowe modele, które zostały wytrenowane na dużych zbiorach danych. Domyślnie, Tesseract 4.0 wykorzystuje model LSTM (ang. Long Short-Term Memory) do rozpoznawania tekstu na obrazach. Aby uzyskać lepsze wyniki w optymalizacji OCR, należy wybrać odpowiedni język, a także dostosować odpowiednie opcje konfiguracyjne Tesseracta.

TESSERACT

```python
try:
    from PIL import Image
except ImportError:
    import Image
import pytesseract

# ścieżka do obrazka z tekstem
image_path = r'[redacted]\image.jpg'

# ustawienia konfiguracji OCR
custom_config = r'--oem 1 --psm 3'

# odczyt tekstu z obrazka przy użyciu Tesseract'a
text = pytesseract.image_to_string(Image.open(image_path), config=custom_config, lang='eng')

# wyświetlenie wyniku
print(text)
```

KRAKEN

```
1  from kraken import pageseg
2  from kraken import binarization
3  from kraken import rnn
4  from kraken.lib import dataset
5  from kraken.lib.util import check_url
6
7  # pobieranie danych wyjściowych
8  datadir = 'https://github.com/mittagessen/kraken/raw/master/tests/data'
9  train_eq = check_url(f'{datadir}/00000014.eq.json')
10
11 # załadowanie zbiorów danych
12 with open(train_eq, 'r') as f:
13     data = dataset.build_dataset([train_eq])
14
15 # konfiguracja sieci neuronowej
16 model = rnn.build_model(train=data, network='lstm', layers=4, units=256, dropout=0.1)
17
18 # trenowanie sieci neuronowej
19 model = rnn.train_model(train=data, model=model, hidden=256, learn_rate=0.001, batch_size=3, max_iters=10)
20
21 # zapisanie wyuczonych wag sieci neuronowej
22 model.save_weights('trained_weights.h5')
```

EASYOCR

```python
import easyocr
import json
import os

# Inicjalizacja czytnika z językiem angielskim
reader = easyocr.Reader(['en'])

# Ścieżka do obrazu
image_path = r'sciezka\do\obrazu.jpg'

# Odczytanie tekstu z obrazu przy użyciu OCR
results = reader.readtext(image_path)

# Inicjalizacja listy, która będzie przechowywać wpisy danych
dataset_entries = []

# Iteracja przez wyniki OCR i zapisanie tekstu oraz współrzędnych ramki
for (bbox, text, prob) in results:
    data_entry = {
        'tekst': text,
        'prawdopodobienstwo': prob,
        'ramka_obrysowujaca': bbox
    }
    dataset_entries.append(data_entry)

# Definicja ścieżki do zbioru danych
dataset_path = r'sciezka\do\zbioru_danych.json'

# Zapisanie wpisów danych do pliku JSON do późniejszego wykorzystania w uczeniu maszynowym
with open(dataset_path, 'w') as f:
    json.dump(dataset_entries, f)

# Informacja dla użytkownika
print(f"Wyniki OCR zostały zapisane w {dataset_path}")
```